Bounded transparency for automated inspection in agriculture

Nicole J.J.P. Koenderink\textsuperscript{a,}\textsuperscript{*}, Jeen Broekstra\textsuperscript{a}, Jan L. Top\textsuperscript{a,}\textsuperscript{b}

\textsuperscript{a} Food & Biobased Research, Wageningen UR, P.O. Box 17 6700 AA, Wageningen, The Netherlands
\textsuperscript{b} Faculty of Sciences, Vrije Universiteit Amsterdam, The Netherlands

\section*{A R T I C L E   I N F O}

Article history:
Received 23 July 2009
Received in revised form 6 February 2010
Accepted 19 February 2010

Keywords:
Decision criterion
Transparency
Computer vision
Knowledge system

\section*{A B S T R A C T}

In agriculture, a major challenge is to automate knowledge-intensive tasks. Task-performing software is often opaque, which has a negative impact on a system's adaptability and on the end user's understanding and trust of the system's operation. A more transparent, declarative way of specifying the expert knowledge required in such software is needed.

We argue that a white-box approach is in principle preferred over systems in which the applied expertise is hidden in the system code. Internal transparency makes it easier to adapt the system to new conditions and to diagnose faulty behaviour. At the same time, explicitness comes at a price and is always bounded by practical considerations. Therefore we introduce the notion of \textit{bounded transparency}, implying a balanced decision between transparency and opaqueness. The method proposed in this paper provides a set of pragmatic objectives and decision criteria to decide on each level of a task's decomposition whether more transparency is sensible or whether delegation to a black-box component is acceptable.

We apply the proposed method in a real-world case study involving a computer vision application for seedling inspection in horticulture and show how bounded transparency is obtained. We conclude that the proposed method offers structure to the application designer in making substantiated implementation decisions.

\section*{1. Introduction}

In the field of computer vision, one of the challenges is to automate knowledge-intensive tasks, \textit{i.e.} tasks that require deep expert knowledge for successful execution. In particular automated inspection of objects typically encountered in agriculture is not easy to realise. The task is hampered by large in-class variations and complex 3D-morphology of the objects, as well as subtle arguments of experts. We submit that automation of such a complex task requires a model of the objects and their assessment that is \textit{explicit where useful}. In traditional applications, the software written to perform a task is opaque in the sense that the expert knowledge used cannot easily be identified in the code. This works well when the role of the software developer and the domain expert completely overlap and when the developers remain responsible for the application over its entire lifetime for all use cases. However, in practice this is hardly ever the case. The considered domains may be hard to master, new application areas and unforeseen conditions may arise over time. Opaqueness makes it hard to maintain and adapt such a system. Even if explicit code not necessarily always produces the best possible inspection results, internal transparency \textit{does} make it easier to adapt the system to new conditions and to diagnose faulty behaviour. At the same time, explicitness comes at a price and is always bounded by practical considerations. Therefore we introduce the notion of \textit{bounded transparency}, implying deliberate use of explicit knowledge at code level. For example, for computing a least-squares fit we generally do not need to understand the detailed inner workings of the algorithm. We can safely rely on a closed procedure obtained from an established numerical library. We only need to understand the necessary control and input-output parameters. However, if the algorithm were made to handle specifically certain features of for example tomato plants, it would be necessary for plant experts to understand how this is done. In this paper we show how bounded transparency can be achieved in computer vision in agriculture by supplementing the use of ontologies with knowledge rules. We do this with an application in inspection of young tomato plants, but we claim that the approach is valid for horticultural or even biological objects in general. At the same time we provide guidelines stating what level of transparency is feasible and sensible in practice. We focus on the following question: \textit{How can we implement bounded transparency in knowledge-intensive computer vision for inspecting biological objects?}

From a pragmatic, business-oriented point of view, organisations introduce automated systems to have tasks performed by them autonomously. If the task is performed well in all expected situations, the user and the organisation do no bother about the
In this paper, we refer to this design approach as black-box. Nowadays this translation may even occur semi-automatically. The advantage is that trade-offs between functional and non-functional requirements can be taken into account at development time. In this type of system development, coding is mostly done in imperative programming languages such as C or Java. This is also typical for industrial computer vision systems where performance is often an issue. In this paper, we refer to this design approach as black-box.

However, increasingly demanding requirements on computer vision applications lead to more complex and intelligent systems. For such knowledge-intensive applications, a black-box approach has severe drawbacks. First, it is not feasible anymore for the software engineers to master all domain expertise. This makes it hard to develop and maintain systems at the required level of intelligence. Second, due to the flexibility of modern processes, it is not possible to foresee all future applications in advance. Moreover, if complex reasoning patterns are implemented it is required to be able to explain these to the user. This adds to commitment of users, but can also serve an educational purpose. We refer to systems that reveal their inner workings in a way that can be understood by users and domain experts as white-box. Finding a trade-off between black-box and white-box is our goal.

This paper is organised as follows. First, we sketch the context of our work. Next, we set out general objectives that motivate transparency. We develop a number of criteria to support making practical design choices and introduce mechanisms for enhancing transparency. Section 4 describes how our design approach works out in the domain of classifying young tomato plants. In Section 5 we evaluate the results. We show how transparency design decisions support developers end users with respect to their objectives. We conclude in Section 6.

2. Related work

In Koenderink et al. (2006), we discussed the use of expert models to implement computer vision applications, i.e., applying ontologies (Gruber, 2008) to express domain concepts. The lessons learned from that discussion included that no satisfactory method for expressing procedural knowledge exists. This insight has led to the work presented in this paper, where we look at how this procedural knowledge should be expressed in an explicit way and provide a method for balancing the need for transparency against other consideration.

The subject of knowledge-intensive computer vision has been studied quite extensively in the fields of machine vision, machine perception, robotics and cognitive vision, since each of these disciplines deals with complex information that has to be interpreted in real-world terms. Typical applications of knowledge-intensive computer vision are robots that move autonomously in the world, or machines that perform inspection tasks. When a robot or machine interacts with the world, it uses vision knowledge to segment the recorded scene on a low level, an intermediate level with ‘semantic’ object parts,1 and a reasoning level that helps in interpreting and acting (Leibe et al., 2008; Nüchter and Hertzberg, 2008). The knowledge on the intermediate level is frequently specified implicitly (Ponweiser et al., 2005; Vincze et al., 2009). For the context of our paper, however, we focus on computer vision applications that use explicit task knowledge.

In the literature, explicit task knowledge is used on various levels. In some papers, the world is described in ‘interpretable objects’ (Crevier and Lepage, 1997). These objects can be represented in image acquisition terminology, as is the case in Boissard et al. (2008), Bosch et al. (2007), Kragic et al. (2005), and Maillot and Thonnat (2008). An example of such a description is: ‘a white fly has a rectangularity vector of [0.5 0.6 0.8 0.85]’ (Boissard et al., 2008). The rectangularity vector and its parameters are explicit, but are not stated in terms from the domain experts’ domain. Renouf et al. (2007), on the other hand, use domain knowledge formulated in the domain experts’ own terminology. They describe a method to formulate image processing applications that allow inexperienced users to build their applications by interaction with the system. Their approach uses a domain ontology that cooperates closely with an image processing ontology. By identifying relevant terms, the user is guided to independently create a computer vision application. In this application, the user works with explicit descriptive knowledge. The procedural knowledge, however, is opaquely encoded in the software and is not available in an explicit manner. Crowley et al. (2007) have chosen to use case based reasoning for their system that can independently diagnose and repair itself. Cases of repair strategies are used to apply procedural knowledge to a problem; the knowledge itself, though, is embedded in the cases and is not formulated explicitly. The work of Clouard et al. (1999) focuses on automatic generation of image processing programs. The resulting computer vision code does not explicitly reflect the domain expert’s procedural knowledge either. Albusac et al. (2009), on the other hand, use both an explicit description of the domain (traffic situations in a surveillance system) and knowledge rules that are stated in terms of expert knowledge. An example of such a rule is ‘if object is not {person, group of people} \ object speed is {slow, medium} \ sidewalk intersection degree is {very low, low} then situation is normal’. The concepts ‘person’, ‘group of people’, or ‘slow’ are understandable terms for a domain expert. They explicitly stress that allowing the domain expert to understand the system output is very important.

Our paper starts from the point of view that transparency is a desirable property. We argue that a computer vision system that is encoded using recognisable knowledge rules leads to significant advantages over an opaque encoding. At the same time, we realise that explicitness comes at a price. Our approach differs from the approaches mentioned above in that we aim to find a balance between transparency and opaqueness.

3. Implementing transparency

Introducing transparency in an application automatically implies making choices as to where transparency stops. Transparency is a requirement for which the benefits should be balanced against the costs. To get a clear image of the trade-offs between black-box and transparent design, we need to take one step back from technical considerations and focus on underlying design objectives and criteria.

3.1. Design objectives and criteria

Any software system has to comply with a set of functional and non-functional requirements. We highlight the following requirements, since they are relevant for automating knowledge-intensive applications.

- First, the system should be fast and accurate enough. It may seem that transparency always implies loss of performance due to the incurred overhead. However, if enhanced transparency leads to

---

1 Semantic object parts (Leibe et al., 2008; Nüchter and Hertzberg, 2008) are recognised object features that are grouped together based on domain knowledge instead of on appearance alone.
better domain modelling and scope setting, it may contribute to system performance.

- Second, a system should be robust and reliable. It should handle all possibly encountered situations properly and signal cases that fall outside the scope of the system. This requirement asks for an accurately modelled scope of the system.

- Third, the system should be trusted by its end users. Transparency of the system can help end users to build confidence in the application.

Besides affecting these general system properties, transparency also has an effect on secondary tasks associated with the system and its development. We distinguish the following secondary tasks.

- System modification (Crevier and Lepage, 1997; Genesereth and Nilsson, 1987), entailing maintenance, modification, trouble shooting, correction, testing, diagnosis, etc. These tasks aim at sustained system improvement. Task support can be given by tools that detect the cause of an error and point to its location in the code or tools that pinpoint modifications needed for a new application area. In general these tasks can be said to contribute to the objectives corrigibility and adaptability.

- Organisational learning (Dhaliwal and Benbasat, 1990), including explanation, education, discussion, elicitation, externalisation, etc. This refers to all tasks that are performed by the system to lift the level of human expertise in the organisation, summarised as the objective understandability.

Next, we provide a set of decision criteria to help improve transparency, but in a bounded fashion: the criteria help weigh costs and benefits to ensure an ideal “hybrid” of white-box and black-box components.

- Availability of explicit domain expertise. If the expert knowledge required for the component can be modelled explicitly, then the component is eligible for explicit specification. Some tasks are typically implicit and hard to express verbally. For example, it is difficult for people to express how they recognise faces.

- Application range. This factor is related to the scope of the considered software component, both in terms of task and domain. The application range indicates the envisaged future tasks and domains for which it might be applied. A specification of the input and output types that the component covers is asked for.

- Common sense and trivial knowledge. Not all knowledge underlying an application should be made visible. Detailing trivial facts (for example how an average value is computed) clutters the system and makes it less transparent.

- Explanation. In order to support tasks such as diagnosis and education, the system should be able to provide a clear explanation of its reasoning process. It may be necessary to give more information about the underlying knowledge than strictly needed to perform the original task.

- Third party expertise and trust. Often, components developed by a third party can be used as part of the system. System developers may or may not trust the origin of such a component. If one believes that the component performs as promised, no need for further specification exists. For example, many predefined mathematical computing libraries from respected software developing companies have been validated and verified extensively. Trust is a main condition for commercial success in that case. Third party components are often optimised for speed of execution, memory food print and scalability.

The identified criteria are a guideline in deciding on the correct level of transparency. They offer structure in designing knowledge-intensive computer vision applications. With these criteria defined, we focus on how to apply the transparency decisions to the application design.

### 3.2. Realisation

Transparency is related to the separation of expert knowledge from implementation details. To increase transparency in an application, expert knowledge must be made explicit and encoded separately from the code that makes it executable and interactive (user interface). The discipline of software engineering has concluded that declarative programming results in more transparency and reusability than imperative programming (Bratko, 2000; Friedman-Hill, 2003). Object orientation and logical programming are examples of environments that move towards expressing code explicitly in a declarative way. Object oriented software development already employs declarative descriptions of classes and attributes of classes. Moreover, rule-based systems are available to express inferences in a logical way.

We distinguish three patterns for making conceptual knowledge explicit in software components (Fig. 1):

- **Task and domain decomposition.** Task decomposition is a technique that is used in many different areas, such as project management (Do, 1982; Martyn, 1975), building projects (Lapp and Golay, 1997), and modularisation in software engineering (van Vliet, 1993). A frequently used method is hierarchical task analysis that breaks down the operation at hand into subtasks (Annett et al., 1971). In creating transparency, it is not so much the decomposition of the activity that is important, but the associated identification on intermediate concepts and relations. Without decomposition, these would remain hidden. The CommonKADS methodology (Schreiber et al., 2000) has formalised this type of decomposition as a way to model knowledge-intensive tasks. At the lowest level of task decomposition it links domain models to the inputs and outputs of so-called inferences. An inference structure is a control-free representation of the information flow.

- **Conceptual refinement.** The facts that form the inputs and outputs of software components can be expressed as instances of a domain ontology. Ontologies allow developers to express internal system facts in terms of domain concepts, at different phases in the reasoning process. Moreover, their taxonomic character assists in specifying this information as precisely as possible. For example, rather than stating that a vision application is intended for classifying small plants, we can state that is meant for the family of Solanaceae (nightshade) only. Besides facts that are used for execution of the task, facts that express the conditions under which the component can be applied also provide transparency.

- **Logical inferences.** Knowledge is essentially a mechanism to derive new facts from existing facts. Logical rules describe the relations between the facts specified in the other two steps. They provide additional insight on how a task is performed, but without resorting to imperative coding.
Note that these ways to add transparency to an application lead to a separation between knowledge description and execution. In the imperative approach, that we have labeled as black-box, a compiler translates procedural expressions to executable code. In this case, the programming code expresses knowledge and execution control at the same time. This introduces dependency between statements and thus obscures the knowledge that is expressed. On the other hand, declarative approaches apply a generic inference engine. This engine can process any set of knowledge rules and leaves control issues hidden to the developer.

In our opinion, the three methods listed are useful means to add transparency to an application. With the first two methods, descriptive domain knowledge can be explicated. The ontologies in which the domain knowledge is modelled are made more specific. With the last method, procedural knowledge is made transparent. By defining procedural knowledge in the form of knowledge rules, the relations between consecutive models of an object are made transparent. In this way, both descriptive and procedural expert knowledge can be embedded in the computer vision design.

The five decision criteria proposed in the previous section – explicit expertise, application range, trivial knowledge, explainability and third-party expertise – are set up to support the designers of a computer vision application to decide on which parts of the application should be defined transparently. These criteria are key to the identified transparency mechanisms in the following way:

The criteria on explicit expertise, trivial knowledge and third-party expertise can lead to a further task and domain decomposition. The explicit expertise criterion investigates whether it is at all possible to express the knowledge in a component in an explicit way. When this criterion is not met, the decision for the component is easy: it has to remain a black-box. The criteria concerning trivial knowledge and third party expertise decide when it is necessary to decompose a task: if the component under inspection is not sufficiently primitive, adding transparency could make sense; if a trusted third-party component is available, then further decomposing of the task may not be necessary.

Conceptual refinement is required based on the application range criterion. This criterion ensures that if a component is only suitable for a limited set of input objects, this fact is stored with the component. By setting conditions on the input objects, we prevent the component from accidentally be used for other objects.

Adding transparency by explicitly defining logical inferences is done based on the trivial knowledge, explainability and third-party expertise criteria. The trivial knowledge criterion indicates that making too detailed knowledge explicit leads to cluttering. The explainability criterion indicates that the explication of knowledge contributes to gaining insight in the inner workings of the component. The third party expertise criterion indicates that existing trusted components can be reused and need not be specified. The combination of these three components allows us to decide on the necessity for formulating procedural knowledge in a declarative format.

4. Case study

In this section, we apply the decision criteria and the means to add transparency introduced in the previous section to a case study in horticulture. The case study is centered around seedling inspection in horticulture in the Netherlands. This task is performed to predict, in an early stage, the amount of high quality fruits or vegetables that a mature plant will produce. The goal is to maximise the probability for a high yield of the whole crop.

Quality inspection of seedlings is a complex task. Seedlings occur in many different shapes and sizes. To predict the yield of a mature plant, a set of heuristically validated quality criteria is used. Some simple criteria concern for example leaf area, stem length, and leaf curvature. Examples of more complex criteria are the likelihood that a plant is a rogue plant, or the irregularity of the leaf shape.

At present, quality assessment is performed by highly-trained experts. They use their knowledge to assess seedlings on all relevant aspects. Even though each expert is intensely trained by the company in which he is employed, the assessment of experts in the same company will typically differ up to ten percent, the inter-expert variation between companies is even higher (Naktuinbouw, 1999). This is in part caused by the fact that the training of the experts is based on the eight officially defined classification rules specified by Naktuinbouw (1999), whereas from interviews with experts we found that over 60 different quality determination rules are used in practice.

To obtain a consistent quality assessment, the horticultural sector is interested in automating the inspection process using computer vision. The sector expressed the wish that the automated system would perform the quality assessment based on the use of the experts’ inspection rules.

The computer vision application to assess tomato seedlings on their quality is created with several of the mentioned objectives in mind. We are interested in adaptability, since that allows the experts to use the application for other objects, such as bell peppers seedlings, or cabbage seedlings, and for other tasks such as grafting. We are also interested in corrigibility, since that allows the tracking down of errors in the application. Moreover, we are interested in reliability. Whenever a batch of seeds is polluted with seeds from a different species, it is important that this is noticed. Without reliability as property, the different seedlings would simply be rejected as ‘abnormal plants’, and would not be identified as plants outside the scope. Also understandability and building trust are desired objectives. The horticultural sector is interested in an application that is trusted by all companies and that can be used to train new personnel.

After recording the object under inspection in the real-world (Koenderink et al., 2009), a point cloud representation of the object is used as input for the computer vision application (Koenderink et al., 2006) (see Fig. 2). At the highest level, the application transforms the 3D recordings of the tomato seedlings into quality classes.

Fig. 2. Each plant is recorded by 12 cameras (4 images are displayed). These images are segmented and are combined to form 3D point clouds. Because of the 3D model, the required level of detail for complex features can be determined.
main task consists of the following steps: (i) a segmentation of the point cloud to obtain a representation of the object in geometric shapes, (ii) an interpretation of the geometric shapes in terms of plant parts, and (iii) a classification of the plant into a set of quality classes (see Fig. 3).

In the following, we do not explicitly specify the entire decision process for this case, but we focus on the first and last of these three steps. From these steps we single out a few components to illustrate the use of the five decision criteria to decide whether more transparency is required for the component. We also show various ways a decision can be implemented. For demonstration purposes, we use a pseudocode representation of the required knowledge rules. The actual implementation takes place using semantic web technologies such as OWL (McGuinness and Van Harmelen, 2004), Jess (Friedman-Hill, 2003). OWL is used to encode ontologies, Jess is used to encode declarative rules. We use Java to deal with the opaque components in the application.

4.1. The segmentation component

The segmentation component has a set of points as input and a set of geometric shapes that are closely connected to plant parts as output. This component is decomposed in a number of subtasks: the creation of point groups, the determining of point types, and the determining of the thick cylinder region. We will zoom in on the first of these subtasks, showing how we apply the decision criteria from Section 3.1.

Creation of point groups. A subtask of the finding of geometric structures is the creation of point groups. For each point \( P \), it results in a point group that has point \( P \) as its central point and that contains all points \( P_i \) that are close enough to \( P \) as neighbours. We want to decide whether this component needs further specification or not, by applying our five criteria.

First, we look at the availability of explicit expertise. The knowledge required to make this component work is covered in an explicit manner by the computer vision expert. Hence, the means to create a white-box expression are available. Expressing the knowledge in a white-box fashion does not really contribute to explainability, but does not clutter the code either. We conclude that the decision criteria give no clear indication on whether a white-box or a black-box expression would be preferred. In this case, we choose to create a declarative implementation of this component:

\[
\text{If } \exists \text{ point } P, \text{ then } \exists \text{ point group } PG \text{ and } PG \text{ has-central-point } P.
\]

For the second subtask, ‘find neighbouring points’, the distance between points has to be calculated. This functionality is available as a trusted third party tool and can be reused. The actual assigning of points to a point group is based on a domain-specific threshold value \( t \) on the distance. We therefore specify it in a declarative fashion with a call to the black-box component ‘distance’:

\[
\begin{align*}
&\exists \text{point group } PG \\
&\text{and } PG \text{ has-central-point } P_c \\
&\text{and } P_1 \text{ is a point and } P_1 \neq P_c \\
&\text{and } \text{distance}(P_1, P_c) < t, \\
&\text{then } PG \text{ has-neighbouring-point } P_1.
\end{align*}
\]

4.2. The classification subtask

The classification subtask has as input a plant model consisting of plant parts and quality parameters, and has as output a quality class that corresponds to the quality of the plant model. In this section, we focus on the component ‘determine quality class’ and decide on the level of detail required for optimal transparency.

Determine quality class. The subtask ‘determine quality class’ decides for an input plant model what the corresponding quality class is. The manner in which this subtask operates is of interest for understandability; the need for explainability is strong. Explicit expertise on how the decision is made is accessible from interviews with domain experts, so explicit expression of knowledge is not inhibited. To further specify the application range, it is important to have insight in the precise decision rules. Different subsets can be used for different tasks or domains. Based on these criteria, we decide to further decompose this task into subtasks:

- \( \text{stem length decision} \)
- \( \text{stem thickness decision} \)
- \( \text{true leaf area decision} \)
- \( \text{combination of individual decisions (see Fig. 5)} \)

Fig. 5. A graphical representation of the specification of the ‘determine quality class’-component.

2 For conciseness reasons, we give only a subset of the classification subtasks.

The ‘stem length decision’ indicates for which stem length a plant is still a candidate for a first class assessment, for which only second class is obtainable, and when a plant should be regarded as an abnormal plant. This rule is valid for all plants and tasks for which stem length is an important quality criterion. The stem length is compared to the average stem length \( k \) of all seedlings. We express this component as a set of three declarative knowledge rules for optimal support of explainability and adaptability:

\[
\begin{align*}
&\text{If plant } P \text{ and } P \text{ has-stem-length } sl \text{ and } sl > 0.8 \ast k, \text{then } P \text{ has-quality-class ‘first choice’}. \\
&\text{If plant } P \text{ and } P \text{ has-stem-length } sl \text{ and } 0.5 \ast k \leq sl \leq 0.8 \ast k, \text{then } P \text{ has-quality-class ‘second choice’}. \\
&\text{If plant } P \text{ and } P \text{ has-stem-length } sl \text{ and } sl < 0.5 \ast k, \text{then } P \text{ has-quality-class ‘abnormal’}. \\
\end{align*}
\]

For the same reasons as above, the ‘stem thickness decision’ is expressed as a declarative rule as well. This rule is valid whenever stem thickness plays a role in the quality assessment of the plant:

\[
\begin{align*}
&\text{If plant } P \text{ and } P \text{ has-stem-thickness } st \text{ and } st < 1 \text{ mm, then } P \text{ has-minus-point –1}. \\
&\text{If plant } P \text{ and } P \text{ has-stem-thickness } st \text{ and } 0.5 \ast k \leq st \leq 0.8 \ast k \text{ then } P \text{ has-quality-class ‘second choice’}. \\
&\text{If plant } P \text{ and } P \text{ has-stem-thickness } st \text{ and } st < 0.5 \ast k, \text{then } P \text{ has-quality-class ‘abnormal’}. \\
\end{align*}
\]

The decision about the true leaf area of a plant is also expressed declaratively. Again, this rule is only used in domains and tasks where true leaf area is an important quality aspect:

\[
\begin{align*}
&\text{If plant } P \text{ and } P \text{ has true-leaf-area } ta \text{ and } ta > 0.5 \ast l, \text{then } P \text{ has-quality-class ‘first choice’}. \\
&\text{If plant } P \text{ and } P \text{ has true-leaf-area } ta \text{ and } ta \leq 0.5 \ast l, \text{then } P \text{ has-quality-class ‘second choice’}. \\
\end{align*}
\]
The ‘combination of individual decisions’ takes the decisions made by the previous three rules and combines them to find the appropriate class of the plant. The knowledge for this subtask is available in an explicit way, the application range suffices for all foreseeable applications, the combination of quality classes and finding the maximum allowed class is trivial knowledge, explainability is not a very high priority for this component, and no third-party components are available that automatically perform this task. These criteria do not lead to a strong bias for using declarative or imperative knowledge at this point. We decide to implement this subtask in an imperative fashion.

5. Evaluation

In this section, we reflect on the results obtained in the case study. We evaluate how the decision criteria have helped us to meet the transparency objectives of the application. Moreover, we show how some decisions would have been different if the set of objectives changes. Next, we show how the decisions made help to realise the desired objectives for the seedling inspection case.

5.1. Reflection on the case study

For each of the five decision criteria used, we can indicate how they contribute to the five objectives of the case study. This discussion is summarised in Table 1.

The criterion concerning explicit expertise deals with the possibility to express knowledge in a transparent way. By explicitly describing the domain knowledge in terms of objects and properties in the application, adaptability, reliability and understandability are supported. In the case study, we have applied the explicit expertise criterion to all tasks where additional transparency was required.

The criterion concerning application range deals with the input objects for which a component is well suited. By explicit specification of the input objects and their attributes, the application knows for which objects the component can be used; it supports building of trust, adaptability and reliability. For the case study, the application range criterion led to a further decomposition of ‘determine quality class’.

The trivial knowledge and common sense criterion indicates whether decomposing and specifying a component leads to more insight in the component or to more cluttering. This criterion helps to find the right level of decomposition and specification; it supports finding the right level of transparency. For example for the component ‘create point groups’ in the case study, it is strongly related to the transparency objectives. Let us look at a component that is used in the quality assessment process. Suppose that the system would need to perform this task in a black-box fashion. The same argument holds for all non-biased components in our case study.

The explanation criterion is strongly linked to further explaining knowledge rules. Explanation allows for a specification of a component’s knowledge in domain relevant terms. It therefore contributes to the objects of understandability and building trust. In the case study, the component ‘determine quality class’ was decomposed based on explanation. By decomposing this component, the subcomponents are made visible and the objectives are served.

The last criterion deals with the availability of third party expertise and the trust one has in a component created by a third party. Since the makers of the components are trusted, using these components contribute to the desired level of trust of the application. Due to the availability of third party components, several components in our case study were implemented as black-box, for example the PCA analysis (part of the determination of point types), the determining of a thick cylinder region (part of the plant part identification) and combining of individual decisions (part of the quality class determination).

It is interesting to look at the influence of an additional objective on the design process. Suppose that the system would need to perform at high speed. In Table 2, we see that the explicit expertise and third party criteria have an influence on the speed objective.

In the case study, the adding of the speed objective leads for example to a clear advise in the case of ‘initialise point groups’. Based on the explicit expertise criterion, we decide that the component is not suitable for further decomposition. Using explicit expertise would lead to an explicit specification of the knowledge that is not profitable for the other objectives. The declarative specification of the knowledge may influence the speed in a negative way. Therefore, we choose to implement this component in a black-box fashion. The same argument holds for all non-biased components in the original case study.

In the same way, we can look at the influence of removing an objective. Let us look at a component that is used in the quality assessment process: ‘splitted heads decision’. A ‘splitted head’ is a defect in a seedling that shows itself as a plant with two sets of true leaves. The splitted head component can be explained in domain specific rules:

If plant $P$ and $P$ has-number-true-leaves 4 and $P$ has-true-leaf-area $> 0.7 \ast 1$,

then $P$ has-splitted-head-probability ‘true’.

If plant $P$ has-splitted-head-probability ‘true’, then $P$ has-quality-class ‘abnormal’.

The first of these rules is an explicit specification of how the value for the property ‘has-splitted-head-probability’ is determined. If understandability is not an objective, the determination of the split-head probability is treated as a black-box component, not as an explicit knowledge rule.

We can conclude that the objectives of the case study play an important role in design decisions to obtain the desired level of transparency. A different set of objectives leads to different decisions, but each decision can be substantiated by the five criteria specified.

\footnote{This component was not mentioned before for conciseness reasons.}

---

**Table 1**

Overview of how the method criteria defined in Section 3.1 support the five case study objectives identified in Section 4.

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Criteria</th>
<th>Explicit expertise</th>
<th>Application range</th>
<th>Trivial knowledge</th>
<th>Explanation</th>
<th>Third party tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability and robustness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build trust</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adaptation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understandability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If plant $P$ and $P$ has-number-true-leaves 4 and $P$ has-true-leaf-area $> 0.7 \ast 1$,

then $P$ has-splitted-head-probability ‘true’.

If plant $P$ has-splitted-head-probability ‘true’, then $P$ has-quality-class ‘abnormal’.

The first of these rules is an explicit specification of how the value for the property ‘has-splitted-head-probability’ is determined. If understandability is not an objective, the determination of the split-head probability is treated as a black-box component, not as an explicit knowledge rule.

We can conclude that the objectives of the case study play an important role in design decisions to obtain the desired level of transparency. A different set of objectives leads to different decisions, but each decision can be substantiated by the five criteria specified.

---

Table 2
Relation between the criteria and a newly defined speed objective.

<table>
<thead>
<tr>
<th></th>
<th>Explicit expertise</th>
<th>Application range</th>
<th>Trivial knowledge</th>
<th>Explanation</th>
<th>Third party tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 6. A flow diagram containing the task knowledge for seedling assessment. With this diagram, new employees can be trained to assess seedlings.

5.2. Design consequences for the transparency objectives

In Section 4, we have made transparency decisions to support the objectives adaptability, corrigibility, reliability, understandability and building trust in our case study. In this section, we look at the practical consequences of the transparency decisions made. Our claim is that due to a correctly defined level of transparency, we can offer tool support to the end user of the application with respect to the desired objectives. In the remainder of this section, we focus on two tools. First, we sketch the possibilities to use the transparent setup of the application to build trust and to support corrigibility. To this end, we propose a tool that allows the end users and the designers to leaf through the application. Second, we show how adaptability and reliability can be supported with a custom-made module that allows end users to add and change assessment rules and reliability criteria.

5.2.1. Leafing through the application

Transparency can be used to build trust, to support corrigibility and to assist in organisational learning. In the design process, the transparency criteria have supported us to design the application such that the components that constitute the application are transparent for the expert to such a degree that they are either explicitly explained, so trivial that they need no explaining, or available as trusted third-party components. The side effect of this design process, is that we can show the inner workings of the application to the end user in just the right amount of detail. By visualising the sequence of models, by pointing out the trusted black-box components and by presenting the declarative rules to the expert in the expert’s own terminology, the expert can gain insight in the steps in the algorithms used and his trust in the application will increase.

With the explicit task knowledge captured in declarative rules and task-specific ontologies, the expert knowledge has been formalised. In the case study this means that experts have indicated which plant features are relevant for assessing the quality of the plants and for which values of these features the plant should be assessed as a first class, second class or abnormal plant. Such explicit task knowledge can easily be represented in a flow diagram that is interpretable by the experts. Such a diagram gives a concise overview of the task knowledge and is useful in training new employees and in discussing about the seedling inspection process. An example of such a flow diagram can be found in Fig. 6. It is a valuable tool for the organisational learning task.

5.2.2. The adaptability and reliability module

The adaptability objective requires that the computer vision application can easily be adapted to changing situations. By specifically indicating for which input objects a component can be used, the expert can identify which components should be adapted when a different set of objects or a different task is asked for.

To allow end users of the application to benefit optimally from the adaptability property, we implement an Adaptation Module. Such an Adaptation Module assists the domain expert to autonomously adapt quality assessment rules according to his needs. Experts can adapt the set of quality rules by changing the decision value of a feature, add or remove a quality class, introduce new quality features or reject features that are already in use. When new features are introduced, the software team obviously has to play a part in implementing the algorithms to calculate the new features. In all other cases, the domain expert should be able to make the changes on his own. A custom-made Adaptation Module can be used to change the quality assessment rules.

We have implemented an Adaptation Module for the case study. The first few lines in the module indicate the quality classes that...
are available in the application.\textsuperscript{4} To change the quality classes that are allowed in the system, new classes can be added, existing classes can be renamed or removed in this part of the Adaptation Module. The next lines contain individual knowledge rules, each corresponding to a component of the procedural knowledge. Each rule is represented with a label, the type of relation, the feature for which the plant is checked, the assessment value, the decision if the feature is below the assessment value, and the decision if the feature is above the assessment value. In the example, the ‘start’ line checks whether the plant is budless. If this is the case, the plant is abnormal, otherwise, the next knowledge rule is to be considered (the default action if no value is specified). To change the decision value of a feature, the domain expert simply has to adapt its value in the Adaptation Module. If an expert wishes to remove a rule, he can delete it from the list of rules. If he is interested in adding an additional rule, he can either use existing features and quality classes or indicate that a new feature is called for. In the last case, the expert can define the quality feature, but has to contact the software team to implement the desired feature. The Adaptation Module allows the expert to compose new rules based on existing building blocks. It gives the flexibility to adapt the quality criteria used to assess the seedlings to meet changing quality standards in practice.

A Reliability Module can be set up as well. The function of this module is to contain relevant exclusion criteria expressed using knowledge rules. There are some nuances to specifying exclusion criteria. Some exclusion rules relate to criteria that are natural to the expert; an example in the case study is to state that a plant can only have one stem, only one plug and maximum four cotyledons. Plants with more cotyledons (such as parsley) or more stems (such as onions) do not meet the criterion. They may occur, but must be recognised as ‘strange plants’. Other rules can be expressed explicitly, but are not naturally a part of the domain expert’s description of the domain. An example is “a plant with leaves connected to the stem where it exits the plug is not a tomato seedling” (but for example a tulip). The occurrence of such objects in the task domain is so unlikely that it seems unnatural to add the corresponding rules to the ontology. Moreover, such exclusion criteria are unnecessary. Users of a computer vision system are interested in recognising and rejecting objects that lie outside the scope. It makes sense to add knowledge rules to recognise foreign objects that may occur in the inspection environment. Bell peppers and other vegetable seedlings are grown in the same environment as tomato seedlings; tulips are not.

6. Conclusion

We have introduced the notion of \textit{bounded transparency} in computer vision for agricultural applications. We argue that a white-box approach is in principle preferred over systems that hide the applied expertise in the programming code. We have proposed decision criteria to decide when transparency is feasible and when delegation to a black-box component is acceptable. We have illustrated the use of the decision criteria in the horticultural domain. These criteria offer structure to the application designer in making substantiated implementation decisions. We have shown that added transparency makes it easier to maintain systems, to diagnose errors and to develop new applications. We have demonstrated that this can even result in improved tool support. The proof-of-principle given in our study is limited to a single case. Nevertheless, we believe that the underlying ideas are sufficiently generic to be extended to other applications. The applicability only depends on the availability of relevant domain knowledge, not on its specific content. However, some limitations of the presented method can already be identified. When considering available expert knowledge we assume that experts are motivated and able to provide this knowledge (in a timely manner). This is not always the case. Additional guidelines on knowledge acquisition can for example be found in the CommonKADS method (Schreiber \textit{et al.}, 2000). Secondly, our method assumes that an appropriate domain decomposition should be made. Decomposition may for example depend on which external software components are available. How this can be done most efficiently is still a subject of research. In spite of these limitations, this case study has shown that the proposed approach is valuable. A deliberate shift from black-box to white-box computer vision in agricultural applications will eventually improve end user experience and development efficacy.

\textbf{Acknowledgements}

We carried out this work within the context of the Virtual Laboratory for e-Science project (www.vl-e.nl), in the Food Informatics subprogram. The project was supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science. The Dutch Ministry of Agriculture provided additional funding.

\textbf{References}


\textsuperscript{4} In a later version of the Adaptation Module, this information can be encoded separately from the knowledge rules to make the module more user-friendly.


